CHAPTER 4

Contents Previous Next

CHAPTER 4 *Regridding Data*

4.1 Overview

CDMS provides several methods for interpolating gridded data:

- from one rectangular, lat−lon grid to another (CDMS regridder)
- between any two lat−lon grids (SCRIP regridder)
- from one set of pressure levels to another
- from one vertical (lat/level) cross−section to another vertical cross−section.

4.1.1 CDMS horizontal regridr

The simplest method to regrid a variable from one rectangular, lat/lon grid to another is to use the **regrid** function defined for variables. This function takes the target grid as an argument, and returns the variable regridded to the target grid:

```
>>> import cdms
>>> f = cdms.open('/pcmdi/cdms/exp/cmip2/ccc/perturb.xml')
>>> rlsf = f('rls') # Read the data
>>> rlsf.shape
(4, 48, 96)

>>> g = cdms.open('/pcmdi/cdms/exp/cmip2/mri/perturb.xml')
>>> rlsg = g['rls'] # Get the file variable (no data read)
>>> outgrid = rlsg.getGrid() # Get the target grid
>>> rlsnew = rlsf.regrid(outgrid)
>>> rlsnew.shape
(4, 46, 72)
>>> outgrid.shape
(46, 72)
```

A somewhat more efficient method is to create a regridder function. This has the advantage that the mapping is created only once and can be used for multiple arrays. Also, this method can be used with data in the form of an MA.MaskedArray or Numeric array. The steps in this process are:

- Given an input grid and output grid, generate a regridder function.
- Call the regridder function on a Numeric array, resulting in an array defined on the output grid. The regridder function can be called with any array or variable defined on the input grid.

The following example illustrates this process. The regridder function is generated at line 9, and the regridding is performed at line 10:

```
1 #!/usr/bin/env python
2 import cdms
```

```
3 from regrid import Regridder
4 f = cdms.open('/pcmdi/cdms/exp/cmip2/ccc/perturb.xml')
5 rlsf = f['rls']
6 ingrid = rlsf.getGrid()
7 g = cdms.open('/pcmdi/cdms/exp/cmip2/mri/perturb.xml')
8 outgrid = g['rls'].getGrid()
9 regridfunc = Regridder(ingrid, outgrid)

10 rlsnew = regridfunc(rlsf)
11 f.close()
12 g.close()
```

Line Notes

2 Makes the CDMS module available.

3 Makes the **Regridder** class available from the **regrid** module.

4 Opens the input dataset.

5 Gets the variable object named '**rls**'. No data is read.

6 Gets the input grid.

7 Opens a dataset to retrieve the output grid.

8 The output grid is the grid associated with the variable named '**rls'** in dataset **g**. Just the grid is retrieved, not the data.

9 Generates a regridder function **regridfunc**.

10 Reads all data for variable **rlsf**, and calls the regridder function on that data, resulting in a transient variable **rlsnew**.

4.1.2 SCRIP horizontal regridder

To interpolate between grids where one or both grids is non−rectangular, CDMS provides an interface to the SCRIP regridder package developed at Los Alamos National Laboratory (http://climate.lanl.gov/Software/SCRIP). Figure 3 illustrates the process:

- Obtain or generate the source and target grids in SCRIP netCDF format. A CDMS grid can be written to a netCDF file, in SCRIP format, using the **write−ScripGrid** method.
- Edit the input namelist file **scrip_in** to reference the grids and select the method of interpolation, either conservative, bilinear, bicubic, or distance−weighted. See the SCRIP documentation for detailed instructions.
- Run the scrip executable to generate a remapping file containing the transformation coefficients.
- CDMS, open the remapping file and create a regridder function with the **readRegridder** method.
- Call the regridder function on the input variable, defined on the source grid. The return value is the variable interpolated to the new grid. Note that the variable may have more than two dimensions. Also note that the input arguments to the regridder function depend on the type of regridder. For example, the bicubic interpolation has additional arguments for the gradients of the variable.

FIGURE 3. Regridding data with SCRIP

Example: Regrid data from a T42 to POP4/3 grid, using the first−order, conservative interpolator.

In this example:

- The input grid is defined in remap_grid_T42.nc.
- The output grid is defined in remap_grid_POP43.nc.
- The input data is variable src_array in file sampleT42Grid.nc.
- The file **scrip_in** has contents:

```
&remap_inputs
 num_maps = 1

 grid1_file = 'remap_grid_T42.nc'
          grid2_file = 'remap_grid_POP43.nc'
          interp_file1 = 'rmp_T42_to_POP43_conserv.nc'
          interp_file2 = 'rmp_POP43_to_T42_conserv.nc'
          map1_name = 'T42 to POP43 Conservative Mapping'
          map2_name = 'POP43 to T42 Conservative Mapping'
          map_method = 'conservative'
          normalize_opt = 'frac'
          output_opt = 'scrip'
          restrict_type = 'latitude'
          num_srch_bins = 90
          luse_grid1_area = .false.
          luse_grid2_area = .false.
/
```

num_maps specifies the number of mappings generated, either 1 or 2. For a single mapping, **grid1_file** and **grid2_file** are the source and target grid definitions, respectively. The **map_method** specifies the type of interpolation, either **'conservative**', **'bilinear**', **'bicubic'**, or **'distwgt'** (distanceweighted). The remaining parameters are described in the SCRIP documentation.

Once the grids and input file are defined, run the scrip executable to generate the remapping file **'rmp_T42_to_POP43_conserv.nc**'

```
% scrip
 Using latitude bins to restrict search.
  Computing remappings between:
 T42 Gaussian Grid
                              and
  POP 4/3 Displaced−Pole T grid
  grid1 sweep
  grid2 sweep
  Total number of links = 63112
```

Next, run CDAT and create the regridder:

```
# Import regrid package for regridder functions
import regrid, cdms
# Read the regridder from the remapper file
```

```
remapf = cdms.open('rmp_T42_to_POP43_conserv.nc')
regridf = regrid.readRegridder(remapf)
remapf.close()
```

Then read the input data and regrid:

```
# Get the source variable
f = cdms.open('sampleT42Grid.nc')
t42dat = f('src_array')
f.close()
# Regrid the source variable
popdat = regridf(dat)
```

Note that **t42dat** can have rank greater than 2. The trailing dimensions must match the input grid shape. For example, if **t42dat** has shape (12, 64, 128), then the input grid must have shape (64,128). Similarly if the variable had a generic grid with shape (8092,), the last dimension of the variable would have length 8092.

4.1.3 Pressure−level regridder

To regrid a variable which is a function of latitude, longitude, pressure level, and (optionally) time to a new set of pressure levels, use the **pressureRegrid** function defined for variables. This function takes an axis representing the target set of pressure levels, and returns a new variable d regridded to that dimension.

```
>>> var.shape
(3, 16, 32)
>>> var.getAxisIds()
['level', 'latitude', 'longitude']
>>> len(levout)
2
>>> result = var.pressureRegrid(levout)
>>> result.shape
(2, 16, 32)
```

4.1.4 Cross−section regridder

To regrid a variable which is a function of latitude, height, and (optionally) time to a new latitude/height cross−section, use the **crossSectionRegridder** defined for variables. This function takes as arguments the new latitudes and heights, and returns the variable regridded to those axes.

```
 >>> varin.shape
(11, 46)
>>> varin.getAxisIds()
['level', 'latitude']
>>> levOut[:]
[ 10., 30., 50., 70., 100., 200., 300., 400., 500.,
   700., 850., 1000.,]
>>> varout = varin.crossSectionRegrid(levOut, latOut)
>>> varout.shape
(12, 64)
```

4.2 regrid module

The regrid module implements the CDMS regridding functionality as well as the SCRIP interface. Although this module is not strictly a part of CDMS, it is designed to work with CDMS objects.

4.2.1 CDMS horizontal regridder

The Python command

from regrid import Regridder

makes the CDMS Regridder class available within a Python program. An instance of Regridder is a function which regrids data from rectangular input to output grids.

Table 4.1 CDMS Regridder Constructor

regridFunction = Regridder(inputGrid, outputGrid)
Create a regridder function which interpolates a data array from input to output grid. Table 4.3 on page 131 describes the calling sequence of this function. *inputGrid* and *outputGrid* are CDMS grid objects. Note: To set the mask associated with *inputGrid* or *outputGrid*, use the grid **setMask** function.

4.2.2 SCRIP Regridder

SCRIP regridder functions are created with the **regrid.readRegridder** function :

Table 4.2 SCRIP Regridder Constructor

regridFunction = regrid.readRegridder(fileobj, mapMethod=None, checkGrid=1)
Read a regridder from an open CDMS file object. *fileobj* is a CDMS file object, as returned from **cdms.open**. *mapMethod* is one of • **'conservative**' : conservative remapper, suitable where area−integrated fields such as water or heat fluxes must be conserved. • **'bilinear'** : bilinear interpolation • **'bicubic**' : bicubic interpolation • **'distwgt'** : distance−weighted interpolation. It is only necessary to specify the map method if it is not defined in the file. If *checkGrid* is 1 (default), the grid cells are checked for convexity, and 'repaired' if necessary. Grid cells may appear to be nonconvex if they cross a 0 / 2pi boundary. The repair consists of shifting the cell vertices to the same side modulo 360 degrees.

4.3 Regridder Functions

It is only necessary to specify the map method if it is not defined in the file.

If *checkGrid* is 1 (default), the grid cells are checked for convexity, and 'repaired' if necessary. Grid cells may appear to be nonconvex if they cross a 0 / 2pi boundary. The repair consists of shifting the cell vertices to the same side modulo 360 degrees.

4.3.1 CDMS regridder functions

A CDMS regridder function is an instance of the CDMS Regridder class. The function is associated with rectangular input and output grids. Typically its use is straightforward: the function is passed an input array and returns the regridded array. However, when the array has missing data, or the input and/or output grids are masked, the logic becomes more complicated.

Step 1: The regridder function first forms an *input mask*. This mask is either two−dimensional or n−dimensional, depending on the rank of the user−supplied mask. If no mask or missing value is specified, the mask is obtained from the data array mask if present.

Two−dimensional case:

- Let *mask_1* be the two−dimensional user mask supplied via the **mask** argument, or the mask of the input grid if no user mask is specified.
- If a missing−data value is specified via the **missing** argument, let the *implicit_mask* be the two−dimensional mask defined as 0 where the first horizontal slice of the input array is missing, 1 elsewhere.
- The input mask is the logical AND(*mask_1*, *implicit_mask*)

N−dimensional case: If the user mask is 3 or 4−dimensional with the same shape as the input array, it is used as the input mask.

Step 2: The data is then regridded. In the two−dimensional case, the input mask is 'broadcast' across the other dimensions of the array. In other words, it assumes that all horizontal slices of the array have the same mask. The result is a new array, defined on the output grid. Optionally, the regridder function can also return an array having the same shape as the output array, defining the fractional area of the output array which overlaps a non−missing input grid cell. This is useful for calculating area−weighted means of masked data.

Step 3: Finally, if the output grid has a mask, it is applied to the result array. Where the output mask is 0, data values are set to the missing data value, or 1.0e20 if undefined. The result array or transient variable will have a mask value of 1 (invalid value) for those output grid cells which completely overlap input grid cells with missing values

Table 4.3 CDMS Regridder function

Type	
Array or Transient−Variable	regridFunction(array, missing=None, order=None,mask=None) Interpolate a gridded data array to a new grid. The interpolation preserves the area−weighted mean on each horizontal slice. If array is a Variable, a TransientVariable of the same rank as the input array is returned, otherwise a masked array is returned. array is a Variable, masked array, or Numeric array of rank 2, 3, or 4. missing is a Float specifying the missing data value. The default is 1.0e20. order is a string indicating the order of dimensions of the array. It has the form returned from variable.getOrder(). For example, the string "tzyx" indicates that the dimension order of array is (time, level, latitude, longitude). If unspecified, the function assumes that the last two dimensions of array match the input grid.

mask is a Numeric array, of datatype Integer or Float, consisting of a fractional number between 0 and 1. A value of 1 or 1.0 indicates that the corresponding data value is to be ignored for purposes of regridding. A value of 0 or 0.0 indicates that the corresponding data value is valid. This is consistent with the convention for masks used by the MA module. A fractional value between 0.0 and 1.0 indicates the fraction of the data value (e.g., the corresponding cell) to be ignored when regridding. This is useful if a variable is regridded first to grid A and then to another grid B; the mask when regridding from A to B would be (1.0 − f) where f is the maskArray returned from the initial grid operation using the returnTuple argument.

If mask is two−dimensional of the same shape as the input grid, it overrides the mask of the input grid. If the mask has more than two dimensions, it must have the same shape as array. In this case, the missing data value is also ignored. Such an ndimensional mask is useful if the pattern of missing data varies with level (e.g., ocean data) or time.

Note: If neither missing or mask is set, the default mask is obtained from the mask of the array if any.

Array, Array

***regridFunction*(ar, missing=None, order=None,**

mask=None, returnTuple=1)
If called with the optional returnTuple argument equal to 1, the function returns a tuple (*dataArray*, *maskArray*). *dataArray* is the result data array. *maskArray* is a Float32 array of the same shape as *dataArray*, such that *maskArray*[i,j] is fraction of the output grid cell [i,j] overlapping a non−missing cell of the grid.

4.3.2 SCRIP Regridder functions

A SCRIP regridder function is an instance of the ScripRegridder class. Such a function is created by calling the **regrid.readRegridder** method. Typical usage is straightforward:

>>> regridf = regrid.readRegridder(remap_file)
>>> outdat = regridf(indat)

The bicubic regridder takes four arguments:

>>> outdat = regridf(indat, gradlat, gradlon, gradlatlon)

A regridder function also has associated methods to retrieve the following fields:

- Input grid
- Output grid
- Source fraction: the fraction of each source (input) grid cell participating in the interpolation.
- Destination fraction: the fraction of each destination (output) grid cell participating in the interpolation.

In addition, a conservative regridder has the associated grid cell areas for source and target grids.

Table 4.4 SCRIP Regridder functions

Return Type

Array or Transient−Variable	**[conservative, bilinear, and distance−weighted regridders]** ***regridFunction*(array)** Interpolate a gridded data array to a new grid. The return value is the regridded data variable. *array* is a Variable, MaskedArray, or Numeric array. The rank of the array may be greater than the rank of the input grid, in which case the input grid shape must match a trailing portion of the array shape. For example, if the input grid is curvilinear with shape (64,128), the last two dimensions of the array must match. Similarly, if the input grid is generic with shape (2560,), the last dimension of the array must have that length.
Array or Transient−Variable	[bicubic regridders] regridFunction(array, gradientLat, gradientLon, gradientLatLon) Interpolate a gridded data array to a new grid, using a bicubic regridder. The return value is the regridded data variable. array is a Variable, MaskedArray, or Numeric array. The rank of the array may be greater than the rank of the input grid, in which case the input grid shape must match a trailing portion of the array shape. For example, if the input grid is curvilinear with shape (64,128), the last two dimensions of the array must match. Simiarly, if the input grid is generic with shape (2560,), the last dimension of the array must have that length. gradientLat: df/di (see the SCRIP documentation). Same shape as array. gradientLon: df/dj. Same shape as array. gradientLatLon: d(df)/(di)(dj). Same shape as array.
Numeric array	getDestinationArea() [conservative regridders only] Return the area of the destination (output) grid cell. The array is 1−D, with length equal to the number of cells in the output grid.
Numeric array	getDestinationFraction() Return the area fraction of the destination (output) grid cell that participates in the regridding. The array is 1−D, with length equal to the number of cells in the output grid.
CurveGrid or Generic−Grid	getInputGrid() Return the input grid, or None if no input grid is associated with the regridder.
CurveGrid or Generic−Grid	getOutputGrid() Return the output grid.

Numeric array	getSourceArea() [conservative regridders only] Return the area of the source (input) grid cell. The array is 1− D, with length equal to the number of cells in the input grid.
Numeric array	getSourceFraction() Return the area fraction of the source (input) grid cell that participates in the regridding. The array is 1−D, with length equal to the number of cells in the input grid.

4.4 Examples

4.4.1 CDMS regridder

Example: Regrid data to a uniform output grid.

```
1  #!/usr/local/bin/python
2  import cdms
3  from regrid import Regridder
4  f = cdms.open('rls_ccc_per.nc')
5  rlsf = f.variables['rls']
6  ingrid = rlsf.getGrid()
7  outgrid = cdms.createUniformGrid(90.0, 46, −4.0, 0.0, 72, 5.0)
8  regridFunc = Regridder(ingrid, outgrid)
9  newrls = regridFunc(rlsf)
10 f.close()
```

Line	Notes
4	Open a netCDF file for input.
7	Create a 4 x 5 degree output grid. Note that this grid is not associated with a file or dataset
8	Create the regridder function
9	Read all data and regrid. The missing data value is obtained from variable rlsf.

Return the area fraction of the source (input) grid cell that participates in the regridding. The array is 1−D, with length equal to the number of cells in the input grid.

Example: Get a mask from a separate file, and set as the input grid mask.

```
1  import cdms
2  from regrid import Regridder
3  #
4  f = cdms.open('so_ccc_per.nc')
5  sof = f.variables['so']
6  ingrid = sof.getGrid()
7  g = cdms.open('rls_mri_per.nc')
8  rlsg = g.variables['rls']
9  outgrid = rlsg.getGrid()
10 regridFunc = Regridder(ingrid,outgrid)
11 h = cdms.open('sft_ccc.nc')
12 sfmaskvar = h.variables['sfmask']
13 sfmask = sfmaskvar[:]
```

```
14 outArray = regridFunc(sof.subSlice(time=0),mask=sfmask)
15 f.close()
16 g.close()
17 h.close()
```

Line Notes

6 Get the input grid.

9 Get the output grid**10 f.close()**

10 Create the regridder function.

13 Get the mask.

14 Regrid with a user mask. The subslice call returns a transient variable corresponding to variable sof at time 0.

Note: Although it cannot be determined from the code, both **mask** and the input array **sof** are four−dimensional. This is the n−dimensional case.

Example: Generate an array of zonal mean values.

```
1  f = cdms.open('rls_ccc_per.nc')
2 rlsf = f.variables['rls']
3 ingrid = rlsf.getGrid()
4 outgrid = cdms.createZonalGrid(ingrid)
5 regridFunc = Regridder(ingrid,outgrid)
6 mean = regridFunc(rlsf)
7 f.close()
```

Line Notes

3 Get the input grid.Return the area fraction of the source (input) grid cell that participates in the regridding. The array is 1−D, with length equal to the number of cells in the input grid.

4 Create a zonal grid. outgrid has the same latitudes as ingrid, and a singleton longitude dimension. **createGlobalMeanGrid** could be used here to generate a global mean array.

5 Generate the regridder function.

6 Generate the zonal mean array.

Example: Regrid an array with missing data, and calculate the area−weighted mean of the result.

```
1   from cdms.MV import *
...
2   outgrid = cdms.createUniformGrid(90.0, 46, −4.0, 0.0, 72, 5.0)
3   outlatw, outlonw = outgrid.getWeights()
4   outweights = outerproduct(outlatw, outlonw)
5   grid = var.getGrid()
```

```
6  sample = var[0,0]
7  latw, lonw = grid.getWeights()
8  weights = outerproduct(latw, lonw)
9  inmask = where(greater(absolute(sample),1.e15),0,1)
10 mean = add.reduce(ravel(inmask*weights*sample))/
   add.reduce(ravel(inmask*weights))
11 regridFunc = Regridder(grid, outgrid)
12 outsample, outmask = regridFunc(sample, mask=inmask,
   returnTuple=1)
13 outmean = add.reduce(ravel(outmask*outweights*outsample))/
   add.reduce(ravel(outmask*outweights))
```

Line	Notes
2	Create a uniform target grid.
3	Get the latitude and longitude weights.
4	Generate a 2−D weights array.
5	Get the input grid. **var** is a 4−D variable.
6	Get the first horizontal slice from **var**.
7−8	Get the input weights, and generate a 2−D weights array.
9	Set the 2−D input mask.
10	Calculate the input array area−weighted mean.
11	Create the regridder function.
12	Regrid. Because returnTuple is set to 1, the result is a tuple (dataArray, maskArray).
13	Calculate the area−weighted mean of the regridded data. mean and outmean should be approximately equal.

4.4.2 SCRIP regridder

Example: Regrid from a curvilinear to a generic grid, using a conservative remapping. Compute the area−weighted means on input and output for comparison.

```
import cdms, regrid, MA

# Open the SCRIP remapping file and data file
direc = ''
fremap = cdms.open(direc+'rmp_T42_to_C02562_conserv.nc')
fdat = cdms.open(direc+'sampleT42Grid.nc')

# Input data array
dat = fdat('src_array')

# Read the SCRIP regridder
regridf = regrid.readRegridder(fremap)

# Regrid the variable
outdat = regridf(dat)


# Get the cell area and fraction arrays. Areas are computed only
# for conservative regridding.
```

```
srcfrac = regridf.getSourceFraction()
srcarea = regridf.getSourceArea()
dstfrac = regridf.getDestinationFraction()
dstarea = regridf.getDestinationArea()

# Calculate area−weighted means
inmean = MA.sum(srcfrac*srcarea*MA.ravel(dat)) / MA.sum(srcfrac*srcarea)
outmean = MA.sum(dstfrac*dstarea*MA.ravel(outdat)) / MA.sum(dstfrac*dstarea)
print 'Input mean:', inmean
print 'Output mean:', outmean


fremap.close)
fdat.close()
```

Contents Previous Next